DTU Library

A Fast Mixed-Precision Strategy for Iterative Gpu-Based Solution of the Laplace Equation

Glimberg, Stefan Lemvig; Engsig-Karup, Allan Peter

Publication date:
2011

Document Version
Publisher's PDF, also known as Version of record

Citation (APA):
Glimberg, S. L. (Author), & Engsig-Karup, A. P. (Author). (2011). A Fast Mixed-Precision Strategy for Iterative Gpu-Based Solution of the Laplace Equation. Sound/Visual production (digital) http://www.siam.org/meetings/pd11/

A Fast Mixed-precision Strategy for Iterative GPU-based Solution of the Laplace Equation

Stefan L. Glimberg

Section of Scientific Computing
Department of Informatics and Mathematical Modelling
Technical University of Denmark

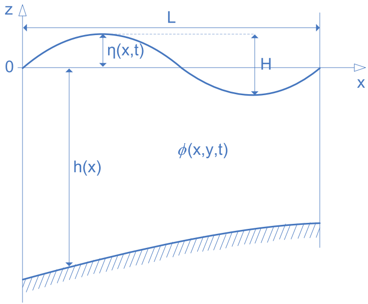
SIAM
Analysis of Partial Differential Equations
November 14^{th}, 2011

A Fast Mixed-precision Strategy for Iterative GPU-based Solution of the Laplace Equation

Fully Nonlinear Free Surface Water Waves

The potential flow equations describe fully nonlinear water waves under the assumption of inviscid and irrotational flow.

2D Potential Flow Equations



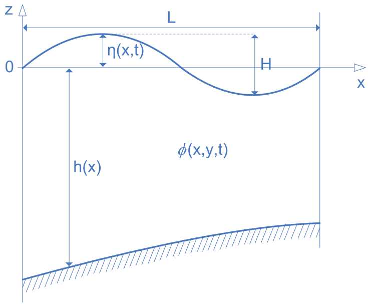
Wave parameters

- η - surface elevation
- ϕ - potential ($u = \nabla\phi$)
- h - still water depth
- $k = 2\pi/L$ - wave number
- kh - dispersion
- H/L - nonlinearity

Fully Nonlinear Free Surface Water Waves

The potential flow equations describe fully nonlinear water waves under the assumption of inviscid and irrotational flow.

2D Potential Flow Equations



$$\partial_t \eta = -\partial_x \eta \, \partial_x \tilde{\phi} + \tilde{\omega}(1 + (\partial_x \eta)^2)$$

$$\partial_t \tilde{\phi} = -g\eta - \frac{1}{2}((\partial_x \tilde{\phi})^2 - \tilde{\omega}^2(1 + (\partial_x \eta)^2))$$

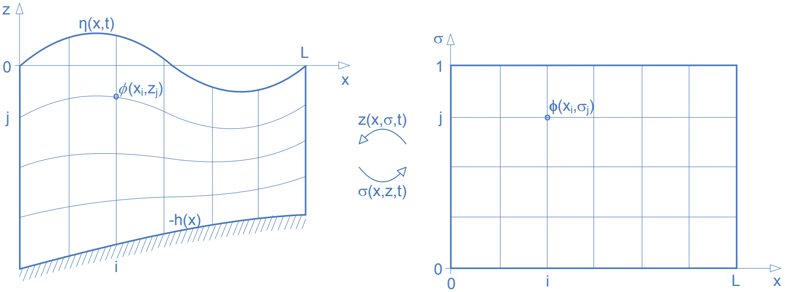
$$\tilde{\omega} = \partial_z \tilde{\phi}, \quad \tilde{\phi} = \phi|_{z=\eta}$$

For $\tilde{\omega}$ to be computed, we need to know the potential in the entire domain.

$$\begin{aligned} \phi &= \tilde{\phi}, & z &= \eta \\ \partial_{xx}\phi + \partial_{zz}\phi &= 0, & -h &\leq z < \eta \\ \partial_z \phi + \partial_x h \, \partial_x \phi &= 0, & z &= -h \end{aligned}$$

σ-Transformed Laplace Equation

$$\sigma(x,z,t) = \frac{z + h(x)}{\eta(x,t) + h(x)}$$



$$\Phi = \tilde{\phi}, \quad \sigma = 1$$

$$\partial_{xx}\Phi + \partial_{xx}\sigma(\partial_\sigma\Phi) + 2\partial_x\sigma(\partial_{x\sigma}\Phi) + ((\partial_x\sigma)^2 + (\partial_z\sigma)^2)\partial_{\sigma\sigma}\Phi = 0, \quad 0 \le \sigma < 1$$

$$(\partial_z\sigma + \partial_x h\partial_x\sigma)\partial_\sigma\Phi + \partial_x h\partial_x\Phi = 0, \quad \sigma = 0$$

Linear Free Surface Water Waves

If wave amplitudes are small $\eta < \epsilon$, then the total water depth is almost the same as the still water depth ($\eta + h \approx h$). If also the derivatives in η and h are assumed to be zero, the free surface equations take linear form.

Linearized Laplace Equation

$$
\begin{aligned}
\Phi = \tilde{\phi}, &\quad \sigma = 1 \\
\partial_{xx}\Phi + (\partial_z\sigma)^2\partial_{\sigma\sigma}\Phi = 0, &\quad 0 \leq \sigma < 1 \\
\partial_z\sigma\,\partial_\sigma\Phi = 0, &\quad \sigma = 0
\end{aligned}
$$

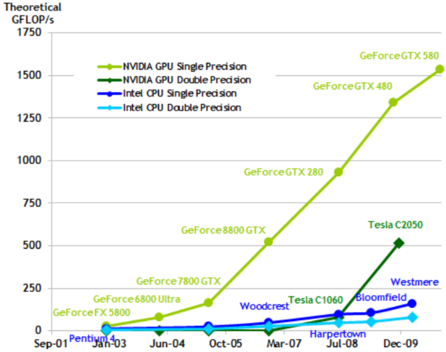
These equations might serve as an approximation for the fully nonlinear equations and can thus be used for preconditioning.

A Fast Mixed-precision Strategy for Iterative GPU-based Solution of the Laplace Equation

Motivation for GPU computing

There are several good reasons to consider Graphical Processing Units for high-performance computing

- Massively parallel architecture, ~ 500 cores.
- Teraflops of floating point performance
- Moderate prices $\$100 - \$2,000$. A personal super computer
- Fairly easy to get started (CUDA, OpenCL)
- Number 2 and 4 on top500 are based on GPUs



A GPU-based Framework for PDE Solvers

We have build a highly generic heterogenous CPU-GPU framework for fast PDE solver prototyping.

Framework Objectives

- Remove all GPU-specific code for the non-expert GPU programmer
- While maintaining the possibility to customize code at kernel level

```
gpulab::vector<float,host_memory>     x_h(100,3.f); // Create host vector x, size 100, value 3
gpulab::vector<float,device_memory>   x_d(x_h);     // Create device vector x, transfer host data
gpulab::vector<float,device_memory>   y_d(x_d);     // Create device vector y, copy device data
y_d.axpy(4.f,x_d);                                  // Do y = a*x+y on the device
y_d.nrm2();                                         // Calculate the 2-norm on the device
```

Ideas are based on the C++ standard library, Thrust, and CUSP that exist for GPUs.

Framework Outline

Key library components

- Regular grid objects, 1D, 2D, 3D.

```
grid_dim<int> dim(100,100);                     // 100x100 grid
grid_dim<double> phys0(0.,0.);                  // Domain starts in x=0, y=0
grid_dim<double> phys1(1.,1.);                  // Domain end in x=1, y=1
grid_properties<int,double> grid_props(dim, phys0, phys1);
grid<double,device_memory> u(grid_props);   // Create u
grid<double,device_memory> f(grid_props);   // Create f
```

Framework Outline

Key library components

- Regular grid objects, 1D, 2D, 3D.
- Compact stencil-based *flexible order FD operators*

```
grid_dim<int> dim(100,100);                        // 100x100 grid
grid_dim<double> phys0(0.,0.);                     // Domain starts in x=0, y=0
grid_dim<double> phys1(1.,1.);                     // Domain end in x=1, y=1
grid_properties<int,double> grid_props(dim, phys0, phys1);
grid<double,device_memory> u(grid_props);          // Create u
grid<double,device_memory> f(grid_props);          // Create f

FD::stencil_2d<double> A(2,4);                     // Second order derivative, fourth order accuracy
A.matvec(u,f);                                     // Calculate f = du/dxx + du/dyy
```

Framework Outline

Key library components

- Regular grid objects, 1D, 2D, 3D.
- Compact stencil-based *flexible order FD operators*
- Iterative methods for solving large systems of eqs.

```
grid_dim<int> dim(100,100);                         // 100x100 grid
grid_dim<double> phys0(0.,0.);                      // Domain starts in x=0, y=0
grid_dim<double> phys1(1.,1.);                      // Domain end in x=1, y=1
grid_properties<int,double> grid_props(dim, phys0, phys1);
grid<double,device_memory> u(grid_props);           // Create u
grid<double,device_memory> f(grid_props);           // Create f

FD::stencil_2d<double> A(2,4);                      // Second order derivative, fourth order accuracy
A.matvec(u,f);                                      // Calculate f = du/dxx + du/dyy

monitor m(iter,rtol,atol);                          // Stopping criteria
solvers::cg cg_solver(A,m);                         // Create a CG solver from A
cg_solver.solve(u,f);                               // Solve Au = f
```

Framework Outline

Key library components

- Regular grid objects, 1D, 2D, 3D.
- Compact stencil-based *flexible order FD operators*
- Iterative methods for solving large systems of eqs.
- Effective preconditioning strategies

```
grid_dim<int> dim(100,100);                  // 100x100 grid
grid_dim<double> phys0(0.,0.);               // Domain starts in x=0, y=0
grid_dim<double> phys1(1.,1.);               // Domain end in x=1, y=1
grid_properties<int,double> grid_props(dim, phys0, phys1);
grid<double,device_memory> u(grid_props);    // Create u
grid<double,device_memory> f(grid_props);    // Create f

FD::stencil_2d<double> A(2,4);               // Second order derivative, fourth order accuracy
A.matvec(u,f);                               // Calculate f = du/dxx + du/dyy

monitor m(iter,rtol,atol);                   // Stopping criteria
solvers::cg cg_solver(A,m);                  // Create a CG solver from A
cg_solver.solve(u,f);                        // Solve Au = f

FD::stencil_2d<double> P(2,2);               // Second order derivative, second order accuracy
cg_solver.set_preconditioner(P);             // Add the preconditioner
cg_solver.solve(u,f);                        // Solve PAu = Pf
```

A Fast Mixed-precision Strategy for Iterative GPU-based Solution of the Laplace Equation

Defect Correction Method

We found that the Defect Correction method works well for our Laplace problem

- High-order approximations (accuracy)
- Minimal storage overhead (problem size)
- Minimal global synchronization and reduction steps (parallelizable)
- Effective as GMRES in practice (effective)

Textbook Recipe

Algorithm: DC Method for approximate solution of $Ax = b$

Choose $x^{[0]}$ `/* initial guess */`
$k = 0$
Repeat
  $r^{[k]} = b - Ax^{[k]}$ `/* high order defect */`
  Solve $M\delta^{[k]} = r^{[k]}$ `/* preconditioner */`
  $x^{[k+1]} = x^{[k]} + \delta^{[k]}$ `/* defect correction */`
  $k = k + 1$
Until convergence or $k > k_{max}$

Defect Correction Method

We found that the Defect Correction method works well for our Laplace problem

- High-order approximations (accuracy)
- Minimal storage overhead (problem size)
- Minimal global synchronization and reduction steps (parallelizable)
- Effective as GMRES in practice (effective)

Textbook Recipe

Algorithm: DC Method for approximate solution of $Ax = b$

Choose $x^{[0]}$ `/* initial guess */`
$k = 0$
Repeat
    $r^{[k]} = b - Ax^{[k]}$ `/* high order defect */`
    Solve $M\delta^{[k]} = r^{[k]}$ `/* preconditioner */`
    $x^{[k+1]} = x^{[k]} + \delta^{[k]}$ `/* defect correction */`
    $k = k + 1$
Until convergence or $k > k_{max}$

Analysis of Defect Correction Convergence

Rewriting DC into the form of a stationary iterative method

$$x^{[k+1]} = x^{[k]} + \mathcal{M}^{-1}(b - \mathcal{A}x^{[k]}) \quad (1)$$

$$= (1 - \mathcal{M}^{-1}\mathcal{A})x^{[k]} + \mathcal{M}^{-1}b \quad (2)$$

$$= \mathcal{G}x^{[k]} + c, \quad k = 0, 1, \ldots \quad (3)$$

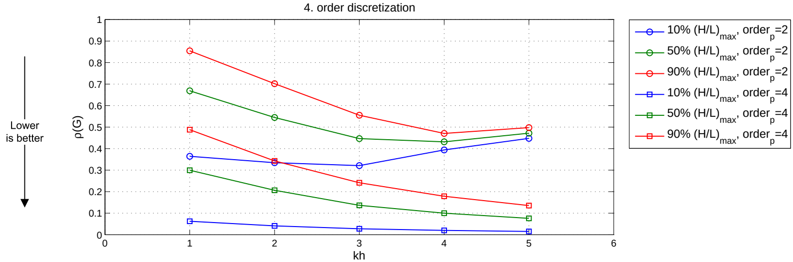
where $\mathcal{G}$ is called the iteration matrix. From stationary iterative theory we know that to ensure convergence towards the exact solution we must have

$$\rho(\mathcal{G}) < 1,$$

where $\rho(\mathcal{G})$ is the spectral radius of $\mathcal{G}$, i.e. the maximum absolute eigenvalue of $\mathcal{G}$. Closer to 0 means better convergence.

Analysis of Defect Correction Convergence

We can now predict attainable convergence rates for various free surface setups using linear flexible-order preconditioners.



Dispersion (kh) expresses ratio between water depth and wave length and relates to the condition number of the Laplacian matrix.

A Fast Mixed-precision Strategy for Iterative GPU-based Solution of the Laplace Equation

Mixed Precision

Definition

- An algorithm that mixes different machine precision numbers in its calculations – while maintaining a high precision solution.

Advantages

Bandwith bound

- 1 double = 2 floats = 64 bits
- Less storage - at all levels
- Less bandwith required

Compute bound

- 1 double multiplier $\approx$ 4 float multipliers
- 1 double adder $\approx$ 2 float adder
- On many GPUs 1:8

Mixed Precision

Definition

- An algorithm that mixes different machine precision numbers in its calculations – while maintaining a high precision solution.

Example

- Single precision roundoff error:

$$c = 0.5 + 0.5 + 0.000000004 - 0.000000003 = 1.000000001 = 1_{fl}$$

- Mixed precision fix:

$$\begin{aligned} a &= 0.5 + 0.5 = 1_{fl} \\ b &= 0.000000004 - 0.000000003 = 0.000000001_{fl} \\ c &= a + b = 1.000000001_{dl} \end{aligned}$$

Mixed Precision Defect Correction

The same principle holds for the defect correction update – and all other refinement processes in general.

Mixed Precision DC

Choose $x^{[0]}$

$k = 0$

Repeat

  $r^{[k]} = b - Ax^{[k]}$ `/* Double Precision */`

  Solve $M\delta^{[k]} = r^{[k]}$ `/* Single Precision */`

  $x^{[k+1]} = x^{[k]} + \delta^{[k]}$ `/* Double Precision */`

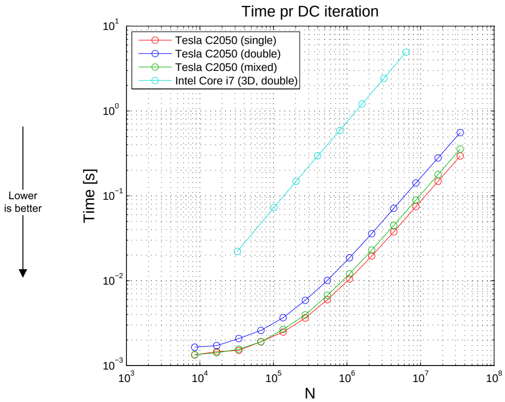
  $k = k + 1$

Until convergence or $k > k_{max}$

Remember, much work lies within the preconditioner!

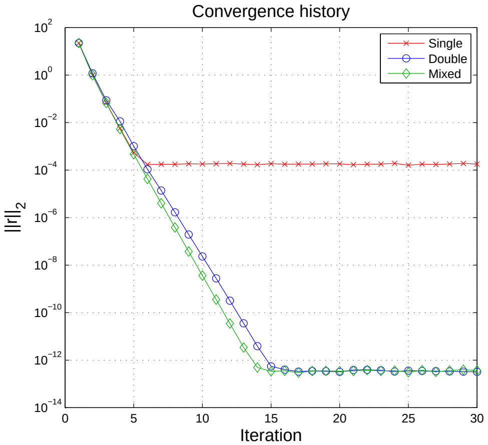
Mixed Precision GPU-based Performance Results

Timings per Defect Correction iteration. Using 6^{th} order accurate stencil, preconditioned with a linear 2^{nd} order accurate multigrid approach, DC+MG-RB-GS-1V(2,2).



Mixed Precision Convergence

The residual error at every iteration confirms that the mixed precision algorithm in fact maintains high accuracy.



A Fast Mixed-precision Strategy for Iterative GPU-based Solution of the Laplace Equation

Allan P. Engsig-Karup.
Efficient low-storage solution of unsteady fully nonlinear water waves using a defect correction method.
Submitted to: Journal of Scientific Computing, 2011.

Allan Peter Engsig-Karup, Morten Gorm Madsen, and Stefan Lemvig Glimberg.
A massively parallel gpu-accelerated model for analysis of fully nonlinear free surface waves.
International Journal for Numerical Methods in Fluids, 2011.